AUTONOMOUS MISSION PLANNING AND SCHEDULING–INNOVATIVE, INTEGRATED, RESPONSIVE

Charisse Sary
Simon Liu
Computer Sciences Corporation

Larry Hull
NASA/Goddard Space Flight Center

Randy Davis
Laboratory for Atmospheric and Space Physics
University of Colorado at Boulder

ABSTRACT

Autonomous mission scheduling, a new concept for NASA ground data systems, is a decentralized and distributed approach to scientific spacecraft planning, scheduling, and command management. Systems and services are provided that enable investigators to operate their own instruments. In autonomous mission scheduling, separate nodes exist for each instrument and one or more operations nodes exist for the spacecraft. Each node is responsible for its own operations which include planning, scheduling, and commanding; and for resolving conflicts with other nodes. One or more database servers accessible to all nodes enable each to share mission and science planning, scheduling, and commanding information. The architecture for autonomous mission scheduling is based upon a realistic mix of state-of-the-art and emerging technology and services, e.g., high performance individual workstations, high speed communications, client-server computing and relational databases. The concept is particularly suited to the smaller, less complex missions of the future.

INTRODUCTION

NASA's scientific spacecraft are unique and valuable resources, so it has always been an important part of mission operations to assure that the time a scientific spacecraft spends in space is utilized as fully as possible in making observations and conducting experiments. To achieve this, most NASA missions plan their scientific activities well in advance; convert those plans into formal spacecraft and instrument schedules on a daily, weekly or monthly basis; and then generate and uplink the commands needed to carry out the scheduled activities.

There are two principal types of mission scheduling problems for NASA. The first type arises when a spacecraft must perform a large number of activities in serial fashion. An example is the Hubble Space Telescope (HST). There are always hundreds of proposed observations in the queue for the HST, and typically only one observation can be made at a time. HST schedulers must select the observations to be supported and then lay them out as single thread of activities. The problem is complicated further by the fact that an experiment may require several observations: if the HST is scheduled to look at a particular target today, then it may also be committed to viewing the target on future occasions as well. Serial scheduling problems are well known (they occur in many terrestrial applications), but they are inherently difficult and time consuming to solve. Developers of automated schedulers for space missions that must handle this kind of problem tend to concentrate on devising

algorithms that increase scheduled observing time while reducing the processing time needed to generate the schedule.

The second type of mission scheduling problem is where a spacecraft can perform a number of major activities in parallel. An example is the forthcoming Earth Observing System (EOS) AM satellite which will carry instruments that can conduct their observing programs simultaneously and more-or-less independently of one another. It has long been recognized that this kind of parallel scheduling problem allows for a distributed solution. Investigators, responsible for each instrument on a spacecraft, generate the schedule for their own instrument. These detailed instrument plans can be collected and combined with a plan for spacecraft housekeeping activities to form a master schedule that can then be checked for conflicts or resource over-subscription.

Since 1986, the Data Systems Technology Division at Goddard Space Flight Center (GSFC) has been investigating scheduling issues relevant to GSFC missions through analysis, prototyping tasks, and testbeds. Recent work has concentrated on EOS and studies of planning and scheduling in a distributed environment. Because the scheduling of observations by most EOS spacecraft falls into the parallel scheduling category described above, the EOS project decided to sponsor an EOS Planning and Scheduling Testbed project during 1992-1993, to explore issues associated with distributed instrument scheduling.

The EOS Testbed was successful in demonstrating that distributed planning and scheduling is feasible for a project like EOS. Several important problems were discovered, but not resolved, however. For example, it proved difficult to keep all of the nodes' scheduling activities synchronized. The scheduling process required substantial coordination between personnel at all nodes. Even when nodes coordinated there were problems, such as nodes not having the most up-to-date ephemeris data available for use in their scheduling.

An interesting result from the EOS Testbed was that conflicts between instruments were usually best resolved by making the instrument investigators aware of the problem and letting them work it out for themselves. To aid in conflict resolution, it would have been useful for investigators to be able to see schedules for instruments other than their own (a feature that the EOS Testbed did not provide). As the testbed progressed the need for a "central scheduler" became less clear. Ideally, every scheduling node—not just the central scheduler—would have access to all information needed for scheduling, and every node would be able to view the spacecraft schedule and any instrument schedule. The ability to detect constraint violations and conflicts, and the potential to automatically resolve simple conflicts, are important capabilities for a distributed scheduling system. However, these functions need not be implemented within a central scheduler.

An autonomous mission scheduling concept has been developed that may eliminate the problems noted above. As shown in Figure 1, separate nodes exist for each instrument and one or more operations nodes exist for the spacecraft. Central to this concept is one or more databases that make needed information available to all nodes. For example, the most up-to-date ephemeris data is always available in a database. Similarly, all nodes have access, via the database(s), to the most current schedules for the spacecraft and for all instruments. All scheduling system transactions become transfers of information to or from a database, using a standard query language (SQL). The schema of a scheduling database is flexible and easy to modify, so new information can be added as needed.

Along with the database approach, the autonomous mission scheduling concept proposes a client-server architecture for a distributed scheduling system. Services, like resource tracking, conflict detection and

conflict resolution, can be invoked by a scheduling node as needed. Distributed scheduling may be one of the first opportunities to actually apply the client-server architecture to space mission operations.

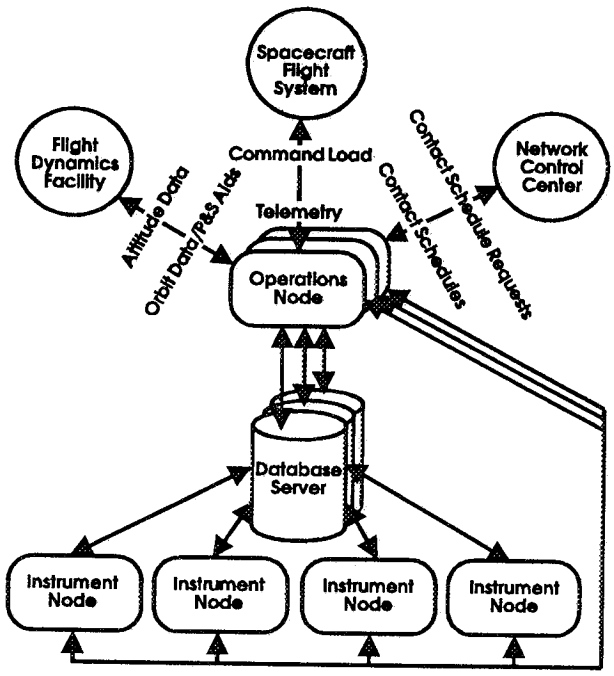

Figure 1. Decentralized and Distributed Scheduling

We believe that, even with the trend toward smaller and simpler spacecraft, distributed scheduling systems may provide new and exciting capabilities. For example, multiple investigators can independently schedule the use of a single shared spacecraft or instrument, or simultaneous observations from multiple spacecraft.

The autonomous mission scheduling operations concept supports key features of the Reusable Network Architecture for Interoperable Space Science, Analysis, Navigation, and Control Environments (Renaissance), a new approach to the development and operation of Mission Operations and Data System Directorate (MO&DSD) ground data systems. This approach avoids technical obsolescence and facilitates hardware and software reuse by using generic components to support science and mission operations. With generic, reusable components, ground data systems will be rapidly and inexpensively built by tailoring components for each new mission. Each ground data system will consist of a number of physically independent, possibly geographically distributed nodes. These nodes would operate together and participate in coordinated planning, scheduling, and commanding using client-server computing and standards-based open systems.

ARCHITECTURE

The autonomous mission scheduling architecture is distributed with application functionality and data partitioned between workstations (clients and servers) connected to local area networks (LANs). Autonomous mission scheduling functions are allocated to components or nodes, and nodes are integrated together to produce a ground system for a target mission. Many different ground system architectures are possible by integrating different combinations of functions and nodes. A typical autonomous mission scheduling architecture is illustrated in Figure 2.

In this architecture, a Mission Operations Center (MOC), the database server, the Flight Dynamics Facility (FDF), and the Network Control Center (NCC) are all located at GSFC. Since a Science Operations Center (SOC) is remote, the MOC and SOC do not share telemetry processing and state vector determination functions. The FDF located at GSFC, provides orbit and attitude planing and scheduling aids. The NCC, located at GSFC, provides network scheduling data to the MOC and remote SOC. A specialized node, a database server, at the MOC, receives and stores this data. Nodes store planning, scheduling, and commanding data on the database server, and may access other nodes' planning, scheduling and commanding data of interest as well. Nodes can access a database

server whether they are remote or not, the only difference being in the kind of network interface used; remote nodes access the database server through a wide area network (WAN) and local nodes through a LAN. The database server node also detects inter-instrument and instrument-spacecraft exceptions, and notifies affected nodes to begin negotiations in order to resolve the exception. GSFC nodes communicate with one another through a LAN, while the remote SOC communicates with GSFC nodes through a WAN.

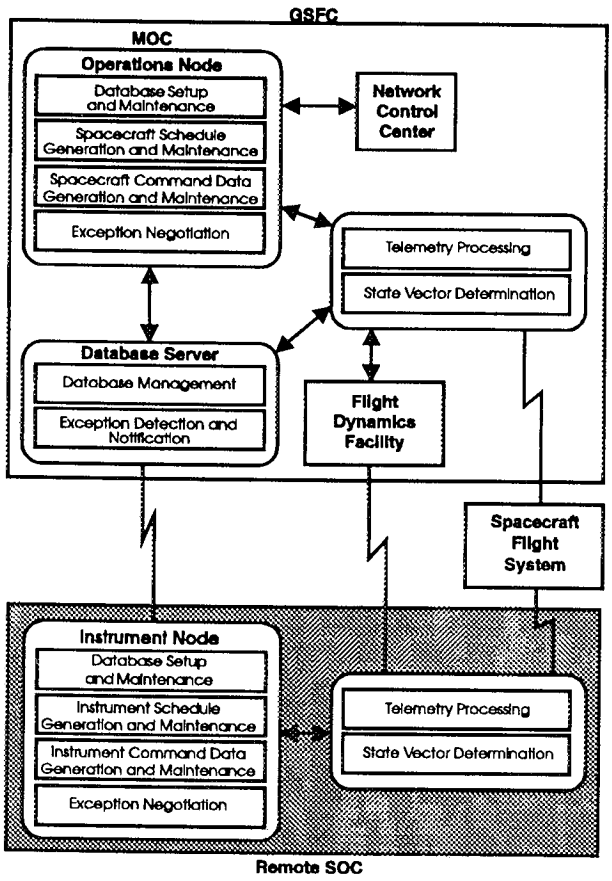

Figure 2. Architecture

The Instrument Node, the Operations Node, and the Database Server Node share several functions. The Database Setup and Maintenance function enables remote or client nodes to access the database server for common planning, scheduling, and commanding data. It stores network schedules, received from the NCC, on the database server, and notifies nodes when this data is initially available. It also maintains the node's local database, which contains data not useful or accessible to other nodes.

The Schedule Generation and Maintenance function generates and stores, on the database server, coordination and operation constraints and activity definitions for the instrument or spacecraft. This information describes nominal operations and planned unique operations and will be used by the database server to detect exceptions later. This function plans and schedules resources to support spacecraft or instrument operations (e.g., scientific observations, calibrations, maintenance), generates and maintains spacecraft or instrument schedules, and stores these schedules on the database server. It designates, as a part of each scheduled activity, the appropriate commands or command sequences to invoke an activity. This function accesses the database server for planning and scheduling data, including data received from the NCC, network resource support schedules, coordination constraints, and activity definitions.

The Command Data Generation and Maintenance function stores instrument or spacecraft command definitions on the database server. Command definitions are used to generate command data and to detect command exceptions. This function extracts the appropriate command or command sequence from command definitions, inserts the necessary parameters, creates the node command data, and stores this data on the database server. It converts composite (instrument and spacecraft) command data to binary, creates a network packet, and uplinks command data to the spacecraft during a Tracking and Data Relay Satellite System (TDRSS) contact. This function also extracts real time command data from the database server, converts command data to uplink format, and uplinks the result when specified to the spacecraft for execution when received onboard. It resolves commanding exceptions,

validates and verifies command data, and maintains command history.

Deviations from normal behavior or unexpected situations are *exceptions*. The Exception Negotiation function coordinates and negotiates with other nodes to resolve exceptions, following the receipt of a message indicating that an exception has occurred.

The Database Management function, provided by a commercial Database Management System (DBMS), manages planning, scheduling, and commanding data stored by nodes. This includes insuring that the data is stored, modified, and accessed correctly, that the security and integrity of the data is maintained, and that distributed, concurrent, reliable, and efficient access is provided.

The Exception Detection and Notification function notifies nodes when new data is available, checks schedules and command data for exceptions, creates a message describing the exception, and forwards the message to affected nodes.

CONCEPT

Long Term Planning

Long term mission planning establishes mission objectives in an overall science operations plan and a long term spacecraft operations plan. Long term mission planning begins with the project scientist and principle investigators producing a long term science plan for the instrument complement. The flight operations team uses this long term plan to develop a corresponding long term plan for spacecraft operation.

With the NASA mission model evolving from a small number of large missions to more numerous but smaller, less complex missions, both the long term science plan and the long term spacecraft operations plan are expected to be relatively brief and to cover largely routine operation, observation, maintenance, and calibration activities. The long term science plan also includes planned, unique operations such as contingency and emergency activities and details concerning coordinated activities and observations.

Based on the long term plan, scientists and flight operators define and store information in the database. The information includes inter-instrument and spacecraft coordination constraints; activity definitions which depict normal operations; command definitions which specify commands, command sequences, and parameters for activity execution; and operation constraints to maintain the health and safety of instruments and spacecraft subsystems.

Initial Scheduling

A large number of instruments have repetitive data acquisition cycles. These natural cycles are not necessarily the same for all instruments on a given mission, and some instruments do not have such cycles, e.g. targeting instruments. Nevertheless, instruments with natural repetitive data acquisition cycles find it easiest to plan and schedule instrument activities within these cycles.

The objective of initial scheduling is to define instrument and spacecraft operation, observation, maintenance, and calibration activities for a given interval. Initial instrument scheduling is done at the SOC and initial spacecraft subsystem scheduling is done at the MOC. All participants in initial scheduling may access available planning and scheduling information in the database. Intra-instrument conflicts are detected and resolved locally at each node. Inter-instrument and instrument-spacecraft conflicts are detected and resolved as described in the next section. The results of initial scheduling are stored in the database.

In the past, for large missions, initial scheduling was used to define requirements for communications resources and services

requested from the NCC. For future smaller missions, the initial schedule will largely be used to detect exceptions. For the less complex missions of the future, requests for communications resources and services are expected to be routine, repetitive, and largely independent of the mission schedule.

Exception Handling

In the past, planning and scheduling systems monitored the scheduling process continuously to detect exceptions. For autonomous mission scheduling, exceptions are detected when the potential arises. An exception does not necessarily have to be an error but is something that requires attention. Exceptions are detected by software and may require special handling. *Exception detection* is checking for and determining that an exception has occurred. *Exception notification* is informing nodes that an exception has occurred. *Exception handling* is responding to a notification and resolving an exception once notified. With this approach, once an exception is detected, it is handled before a major problem arises.

Exceptions can be schedule or command data exceptions. The three types of exceptions are:

- operator errors such as failing to produce information by a deadline or storing incorrect information.
- deviations from normal operations which may or may not be erroneous. An example of a deviation is a late change which is not preplanned and uses leftover available resources. Deviations do not necessarily create conflicts.
- resource, constraint, intra-instrument, and inter-instrument conflicts.

When an event occurs, exception detection is invoked. Two events that trigger exception detection are:

- operator actions such as adding to, deleting from, or updating the database.
- deadlines for performing an action or receiving data such as missing a deadline for receiving an initial schedule.

If an exception is detected, an exception notification message is generated and sent to the nodes involved. If more than one node is involved, one node is given primary authority for resolving the conflict. The responsible node may be:

- The owner of the activity that contributes the most to the conflict.
- The owner of the most critical or most important activity.
- The involved node that has the most restrictive operation constraints.

Upon receiving a notification message, nodes analyze exception data contained within the message, resolve any internal errors, deviations, or conflicts, and negotiate with other nodes, if necessary, to resolve inter-instrument or instrument-spacecraft conflicts. Exception handling, at any node, is expected to be performed manually by an operator or automatically with user agents. Automation will be introduced gradually based on operator need and software maturity. Using exception history, user agents can be developed to handle exceptions that have occurred previously and are likely to recur. A unique user agent is defined for each exception. The initial system automatically handles only a few exceptions and contains only a few user agents. As the system matures, it is expected to handle more exceptions and to contain many user agents.

With user agents, the automation level can change dynamically depending on operator workload, level of expertise, and preference. When an exception occurs, the system automatically invokes the appropriate user

agent to handle the exception. However, operators still have final authority over decisions made. They can override the user agent operating at a node and direct the node to do something different than it would have chosen automatically. Also, if an exception occurs that the system cannot handle, operators become involved. Human operators may want or need to negotiate among themselves to resolve exceptions using the telephone, electronic mail, or other methods.

Final Scheduling

Final scheduling is the last step in the planning and scheduling process. The final schedule is an executable, exception-free, composite schedule of instrument and spacecraft operation, observation, maintenance, and calibration activities for a given time interval. Final scheduling is the process of incorporating the results of the exception handling process, and any changes that have occurred including late changes or targets of opportunity, in the initial schedule. Targets of opportunity are phenomena of interest that cannot be predicted, are often short-lived, or are changing rapidly. As throughout the scheduling process, final instrument scheduling is done at the SOC and final spacecraft subsystem scheduling is done at the MOC. The results of final scheduling are stored in the database where last minute inter-instrument and spacecraft-instrument conflicts can be detected and resolved as described above. Changes are permitted as long as there is ample time to handle them, they do not cause an exception, and they can be accommodated within the communications resources and services obtained from the NCC.

Commanding

The objective of commanding is to direct the spacecraft and instruments to perform scheduled or other required activities. Commanding involves generating, uplinking, storing, and executing command data. There are three major levels of commanding: normal commanding, contingency commanding, and emergency commanding.

Normal commanding directs the spacecraft to perform scheduled spacecraft and instrument activities. Command data is stored in the database so that exceptions can be detected and resolved. When and how often command data is generated varies by mission. Command data is generated from scheduled activities. Each SOC is responsible for its own instrument command generation while the MOC is responsible for spacecraft subsystem command generation. The MOC is responsible for assembling the instrument and spacecraft command data and uplinking the composite command data set to the spacecraft during a communication link.

Spacecraft and instrument constraints are defined prior to launch and stored in the database. The MOC and SOCs validate all spacecraft and instrument command data before it is uplinked by the MOC. They also verify that command data was received onboard completely, correctly, and in sequence, and that command data was stored and executed properly. All onboard command data is verified by evaluating the appropriate return-link housekeeping and engineering parameters. The MOC and SOC maintain their respective command history archives.

Contingency commanding directs the spacecraft to perform contingency spacecraft and instrument activities, possibly due to late changes or targets of opportunity. Since most contingency activities are preplanned, the associated command data can be stored in the database. If no preplanned command data is available, the responsible node must generate the command data in sufficient time so as not to subject the mission to undue risk. When accepted, the schedule is updated, and a new command data set is generated and uplinked at the appropriate time.

Emergency commanding directs the spacecraft to perform spacecraft and instrument safing operations, generally in reaction to some potentially catastrophic event. Emergency commanding for the spacecraft subsystem is performed by the MOC. Emergency commanding for an instrument is performed by the SOC using the results of instrument monitoring. Whenever practical, emergency command data is preplanned and stored in the database for later use. If unavailable, the responsible node generates the command data. When initiated, emergency commands are validated and uplinked at the next available communication link. The responsible node monitors the return-link telemetry to verify the receipt and execution of emergency commands.

FUTURE WORK

We plan to prototype the concept described above, and plan to develop a representative subset of components: a planning and scheduling database at GSFC, a MOC at GSFC, and two SOCs—one at GSFC and one at the University of Colorado (CU). The command management portions of the concept will not be prototyped.

The planning and scheduling database and the CU SOC will be implemented on VAX workstations. The MOC and the GSFC SOC will be implemented on SUN 4 workstations. A commercial DBMS, SYBASE, will be used to implement the database server functionality with all nodes having SYBASE client functionality for distributed access.

The MOC and SOC at GSFC will use an enhanced Request Oriented Scheduling Engine (ROSE) scheduler. The SOC located at CU will use an enhanced Operations and Science Instrument Support Planning and Scheduling (OASIS-PS). ROSE and OASIS-PS are written in Ada and use the Transportable Applications Environment Plus (TAE+) (Century Computing, Inc., 1993) for the user interface.

REFERENCES

Buford, Carolyn. (August 1992). *Scheduling Data Representation: Concept and Experience in Code 520*, (DSTL-92-010). Greenbelt, MD: NASA/Goddard Space Flight Center.

University of Colorado at Boulder, Jet Propulsion Laboratory, Goddard Space Flight Center. (June 1993). *Earth Observing System Distributed Planning and Scheduling Prototype Lessons Learned Working Paper.*

NASA/Goddard Space Flight Center. (April 23, 1993). *Future Mission Operations and Data Systems Directorate Architecture and Ground Segment Operations Concept.* Greenbelt, MD.

Century Computing, Inc. (September 1993). *TAE+ User Interface Developer's Guide, Version 5.3*, Greenbelt, MD: NASA/Goddard Space Flight Center.

Fahnestock, Dale. (November 18, 1993). *Renaissance, A New Approach to Ground Data Systems.* Presentation held at Greenbelt, MD.